Introduction

I want to tell you a story.

No, not the story of how, in 1991, Linus Torvalds wrote the first version of the Linux kernel. You can read that story in lots of Linux books. Nor am I going to tell you the story of how, some years earlier, Richard Stallman began the GNU Project to create a free Unix-like operating system. That's an important story too, but most other Linux books have that one, as well.

No, I want to tell you the story of how you take back control of your computer.

When I began working with computers as a college student in the late 1970s, there was a revolution going on. The invention of the microprocessor had made it possible for ordinary people like you and me to actually own a computer. It's hard for many people today to imagine what the world was like when only big business and big government ran all the computers. Let's just say, you couldn't get much done.

Today, the world is very different. Computers are everywhere, from tiny wristwatches to giant data centers to everything in between. In addition to ubiquitous computers, we also have a ubiquitous network connecting them together. This has created a wondrous new age of personal empowerment and creative freedom, but over the last couple of decades something else has been happening. A few giant corporations have been imposing their control over most of the world's computers and deciding what you can and cannot do with them. Fortunately, people from all over the world are doing something about it. They are fighting to maintain control of their computers by writing their own software. They are building Linux.

Many people speak of “freedom” with regard to Linux, but I don't think most people know what this freedom really means. Freedom is the power to decide what your computer does, and the only way to have this freedom is to know what your computer is doing. Freedom is a computer that is without secrets, one where everything can be known if you care enough to find out.

Why Use the Command Line?

Have you ever noticed in the movies when the “super hacker,”—you know, the guy who can break into the ultra-secure military computer in less than 30 seconds—sits down at the computer, he never touches a mouse? It's because filmmakers realize that we, as human beings, instinctively know the only way to really get anything done on a computer is

by typing on a keyboard!

Most computer users today are familiar only with the *graphical user interface* (GUI) and have been taught by vendors and pundits that the *command line interface* (CLI) is a terrifying thing of the past. This is unfortunate, because a good command line interface is a marvelously expressive way of communicating with a computer in much the same way the written word is for human beings. It's been said that “graphical user interfaces make easy tasks easy, while command line interfaces make difficult tasks possible” and this is still very true today.

Since Linux is modeled after the Unix family of operating systems, it shares the same rich heritage of command line tools as Unix. Unix came into prominence during the early 1980s (although it was first developed a decade earlier), before the widespread adoption of the graphical user interface and, as a result, developed an extensive command line interface instead. In fact, one of the strongest reasons early adopters of Linux chose it over, say, Windows NT was the powerful command line interface that made the “difficult tasks possible.”

What This Book Is About

This book is a broad overview of “living” on the Linux command line. Unlike some books that concentrate on just a single program, such as the shell program, `bash`, this book will try to convey how to get along with the command line interface in a larger sense. How does it all work? What can it do? What's the best way to use it?

This is not a book about Linux system administration. While any serious discussion of the command line will invariably lead to system administration topics, this book touches on only a few administration issues. It will, however, prepare the reader for additional study by providing a solid foundation in the use of the command line, an essential tool for any serious system administration task.

This book is very Linux-centric. Many other books try to broaden their appeal by including other platforms such as generic Unix and macOS. In doing so, they “water down” their content to feature only general topics. This book, on the other hand, only covers contemporary Linux distributions. Ninety-five percent of the content is useful for users of other Unix-like systems, but this book is highly targeted at the modern Linux command line user.

Who Should Read This Book

This book is for new Linux users who have migrated from other platforms. Most likely you are a “power user” of some version of Microsoft Windows. Perhaps your boss has told you to administer a Linux server, or you’re entering the exciting new world of single board computers (SBC) such as the Raspberry Pi. You may just be a desktop user who is tired of all the security problems and wants to give Linux a try. That's fine. All are welcome here.

That being said, there is no shortcut to Linux enlightenment. Learning the command line is challenging and takes real effort. It's not that it's so hard, but rather it's so *vast*. The average Linux system has literally *thousands* of programs you can employ on the command line. Consider yourself warned; learning the command line is not a casual endeavor.

On the other hand, learning the Linux command line is extremely rewarding. If you think you're a “power user” now, just wait. You don't know what real power is—yet. And, unlike many other computer skills, knowledge of the command line is long lasting. The skills learned today will still be useful 10 years from now. The command line has survived the test of time.

It is also assumed that you have no programming experience, but don’t worry, we'll start you down that path as well.

What's in This Book

This material is presented in a carefully chosen sequence, much like a tutor sitting next to you guiding you along. Many authors treat this material in a “systematic” fashion, exhaustively covering each topic in order. This makes sense from a writer’s perspective, but can be very confusing to new users.

Another goal is to acquaint you with the Unix way of thinking, which is different from the Windows way of thinking. Along the way, we'll go on a few side trips to help you understand why certain things work the way they do and how they got that way. Linux is not just a piece of software; it's also a small part of the larger Unix culture, which has its own language and history. I might throw in a rant or two, as well.

This book is divided into four parts, each covering some aspect of the command line experience:

- **Part 1 – Learning The Shell** starts our exploration of the basic language of the command line including such things as the structure of commands, file system navigation, command line editing, and finding help and documentation for commands.
- **Part 2 – Configuration And The Environment** covers editing configuration files that control the computer's operation from the command line.
- **Part 3 – Common Tasks And Essential Tools** explores many of the ordinary tasks that are commonly performed from the command line. Unix-like operating systems, such as Linux, contain many “classic” command line programs that are used to perform powerful operations on data.
- **Part 4 – Writing Shell Scripts** introduces shell programming, an admittedly rudimentary, but easy to learn, technique for automating many common computing tasks. By learning shell programming, you will become familiar with concepts that can be applied to many other programming languages.

How To Read This Book

Start at the beginning of the book and follow it to the end. It isn't written as a reference work, it's really more like a story with a beginning, middle, and end.

Prerequisites

To use this book, all you will need is a working Linux installation. You can get this in one of two ways:

1. **Install Linux on a (not so new) computer.** It doesn't matter which distribution you choose, though most people today start out with either Ubuntu, Fedora, or OpenSUSE. If in doubt, try Ubuntu first. Installing a modern Linux distribution can be ridiculously easy or ridiculously difficult depending on your hardware. I suggest a desktop computer that is a couple of years old and has at least 2 GB of RAM and 6 GB of free hard disk space. Avoid laptops and wireless networks if at all possible, as these are often more difficult to get working.
2. **Use a "Live CD" or USB flash drive.** One of the cool things you can do with many Linux distributions is run them directly from a CDROM or USB flash drive without installing them at all. Just go into your BIOS setup and set your computer to boot from a CDROM drive or USB device, and reboot. Using this method is a great way to test a computer for Linux compatibility prior to installation. The disadvantage is that it may be very slow compared to having Linux installed on your hard drive. Both Ubuntu and Fedora (among others) have live versions.

Regardless of how you install Linux, you will need to have occasional superuser (i.e., administrative) privileges to carry out the lessons in this book.

After you have a working installation, start reading and follow along with your own computer. Most of the material in this book is "hands on," so sit down and get typing!

Why I Don't Call It "GNU/Linux"

In some quarters, it's politically correct to call the Linux operating system the "GNU/Linux operating system." The problem with "Linux" is that there is no completely correct way to name it because it was written by many different people in a vast, distributed development effort. Technically speaking, Linux is the name of the operating system's kernel, nothing more. The kernel is very important of course, since it makes the operating system go, but it's not enough to form a complete operating system.

Enter Richard Stallman, the genius-philosopher who founded the Free Software movement, started the Free Software Foundation, formed the GNU Project, wrote the first version of the GNU C Compiler (gcc), created the GNU General Public

License (the GPL), etc., etc., etc. He *insists* that you call it “GNU/Linux” to properly reflect the contributions of the GNU Project. While the GNU Project predates the Linux kernel, and the project's contributions are extremely deserving of recognition, placing them in the name is unfair to everyone else who made significant contributions. Besides, I think “Linux/GNU” would be more technically accurate since the kernel boots first and everything else runs on top of it.

In popular usage, “Linux” refers to the kernel and all the other free and open source software found in the typical Linux distribution, that is, the entire Linux ecosystem, not just the GNU components. The operating system marketplace seems to prefer one-word names such as DOS, Windows, macOS, Solaris, Irix, and AIX. I have chosen to use the popular format. If, however, you prefer to use “GNU/Linux” instead, please perform a mental search-and-replace while reading this book. I won't mind.

What's New in the Fifth Internet Edition

Building on the work performed for the Fourth Internet Edition, this edition of *The Linux Command Line* has been extensively modernized. There are numerous small edits and corrections, new screenshots and diagrams, along with a few clarifications. I also fixed a couple of bugs ;-).

Acknowledgments

I want to thank the following people, who helped make this book possible:

First Internet Edition

Jenny Watson, Acquisitions Editor at Wiley Publishing who originally suggested that I write a shell scripting book.

John C. Dvorak, noted columnist and pundit. In an episode of his video podcast, “Cranky Geeks,” Mr. Dvorak described the process of writing: “Hell. Write 200 words a day and in a year, you have a novel.” This advice led me to write a page a day until I had a book.

Dmitri Popov wrote an article in Free Software Magazine titled, “Creating a Book Template with Writer,” which inspired me to use OpenOffice.org Writer for composing the text. As it turned out, it worked wonderfully.

Mark Polesky performed an extraordinary review and test of the text.

Jesse Becker, Tomasz Chrzczonowicz, Michael Levin, and Spence Miner also tested and reviewed portions of the text.

Karen M. Shotts contributed a lot of hours, polishing my so-called English by editing the manuscript.

Second Internet Edition

Special thanks go out to the following individuals who provided valuable feedback incorporated into the Second Internet Edition: Adrian Arpidez, Hu Bo, Heriberto Cantú, Joshua Escamilla, Bruce Fowler, Ma Jun, Seth King, Mike O'Donnell, Parviz Rasoulipour, Gabriel Stutzman, and Christian Wuethrich.

Third Internet Edition

Special thanks go out to the following individuals who provided valuable feedback incorporated into the Third Internet Edition: Steve Bragg, Lixin Duan, Sunil Joshi, Chris Knight, Jim Kovacs, Bartłomiej Majka, Bashar Maree, Frank McTipps, Justin Page, Waldo Ribeiro, Satej Kumar Sahu, Mikhail Sizov, Pickles Spill, Francesco Turco, Wolfram Volpi, and Boyang Wang .

Fourth Internet Edition

Special thanks go out to the following individuals who provided valuable feedback incorporated into the Fourth Internet Edition: Enzo Cardinal, Devin Harper, Jørgen Heitmann, Jonathan Jones, Jaroslaw Kolosowski, Eric.Kammerer, Waldo Ribeiro, Nick Rose, Ben Slater, and Francesco Turco.

Fifth Internet Edition

Special thanks go out to the following individuals who provided valuable feedback incorporated into the Fifth Internet Edition: John Burns, Paolo Casati, Waldo Ribeiro, and Valter Wierzba.

And lastly, many thanks to the many readers of LinuxCommand.org, who have sent me so many kind emails. Their encouragement gave me the idea that I was really on to something!

Your Feedback Is Needed!

This book is an ongoing project, like many open source software projects. If you find a technical error, drop me a line at:

bshotts@users.sourceforge.net

Be sure to indicate the exact edition of the book you are reading. Your changes and suggestions may get into future releases.

Further Reading

- Here are some Wikipedia articles about the famous people mentioned above:
  http://en.wikipedia.org/wiki/Linus_Torvalds
  http://en.wikipedia.org/wiki/Richard_Stallman

- The Free Software Foundation and the GNU Project:
  http://en.wikipedia.org/wiki/Free_Software_Foundation
  http://www.fsf.org
  http://www.gnu.org
- Richard Stallman has written extensively on the “GNU/Linux” naming issue:
  http://www.gnu.org/gnu/why-gnu-linux.html
  http://www.gnu.org/gnu/gnu-linux-faq.html#tools

Colophon

This book was originally written using OpenOffice.org Writer in Liberation Serif and Sans fonts on a Dell Inspiron 530N, factory configured with Ubuntu 8.04. The PDF version of the text was generated directly by OpenOffice.org Writer. The Second Internet Edition was produced on the same computer using LibreOffice Writer on Ubuntu 12.04. The Third and Fourth Internet Editions were produced with LibreOffice Writer on a System76 Ratel Pro computer, factory configured with Ubuntu 14.04. The Fifth Internet Edition was produced on the same computer using LibreOffice Writer and Ubuntu 18.04.

Part 1 – Learning the Shell

1 – What Is the Shell?

When we speak of the command line, we are really referring to the *shell*. The shell is a program that takes keyboard commands and passes them to the operating system to carry out. Almost all Linux distributions supply a shell program from the GNU Project called `bash`. The name “bash” is an acronym for “Bourne Again SHell”, a reference to the fact `bash` is an enhanced replacement for `sh`, the original Unix shell program written by Steve Bourne.

Terminal Emulators

When using a graphical user interface (GUI), we need another program called a *terminal emulator* to interact with the shell. If we look through our desktop menus, we will probably find one. KDE uses `konsole` and GNOME uses `gnome-terminal`, though it's likely called simply “terminal” on our menu. A number of other terminal emulators are available for Linux, but they all basically do the same thing; give us access to the shell. You will probably develop a preference for one or another terminal emulator based on the number of bells and whistles it has.

Making Your First Keystrokes

So let's get started. Launch the terminal emulator! Once it comes up, we should see something like this:

```
[me@linuxbox ~]$
```

This is called a *shell prompt* and it will appear whenever the shell is ready to accept input. While it may vary in appearance somewhat depending on the distribution, it will typically include your *username@machinename*, followed by the current working directory (more about that in a little bit) and a dollar sign.

Note: If the last character of the prompt is a pound sign (“#”) rather than a dollar

sign, the terminal session has *superuser* privileges. This means either we are logged in as the root user or we selected a terminal emulator that provides superuser (administrative) privileges.

Assuming things are good so far, let's try some typing. Enter some gibberish at the prompt like so:

```
[me@linuxbox ~]$ kaekfjaeifj
```

Because this command makes no sense, the shell tells us so and give us another chance.

```
bash: kaekfjaeifj: command not found
[me@linuxbox ~]$
```

Command History

If we press the up-arrow key, we will see that the previous command `kaekfjaeifj` reappears after the prompt. This is called *command history*. Most Linux distributions remember the last 1000 commands by default. Press the down-arrow key and the previous command disappears.

Cursor Movement

Recall the previous command by pressing the up-arrow key again. If we try the left and right-arrow keys, we'll see how we can position the cursor anywhere on the command line. This makes editing commands easy.

A Few Words About Mice and Focus

While the shell is all about the keyboard, you can also use a mouse with your terminal emulator. A mechanism built into the X Window System (the underlying engine that makes the GUI go) supports a quick copy and paste technique. If you highlight some text by holding down the left mouse button and dragging the mouse over it (or double clicking on a word), it is copied into a buffer maintained by X. Pressing the middle mouse button will cause the text to be pasted at the cursor location. Try it.

Note: Don't be tempted to use `Ctrl-c` and `Ctrl-v` to perform copy and paste inside a terminal window. They don't work. These control codes have different meanings to the shell and were assigned many years before the release of Microsoft Windows.

Your graphical desktop environment (most likely KDE or GNOME), in an effort to behave like Windows, probably has its *focus policy* set to "click to focus." This means for a window to get focus (become active) you need to click on it. This is contrary to the traditional X behavior of "focus follows mouse" which means that a window gets focus just by passing the mouse over it. The window will not come to the foreground until you click on it but it will be able to receive input. Setting the focus policy to "focus follows mouse" will make the copy and paste technique even more useful. Give it a try if you can (some desktop environments such as Ubuntu's Unity no longer support it). I think if you give it a chance you will prefer it. You will find this setting in the configuration program for your window manager.

Try Some Simple Commands

Now that we have learned to enter text in our terminal emulator, let's try a few simple commands. Let's begin with the `date` command, which displays the current time and date.

```
[me@linuxbox ~]$ date
Thu Mar  8 15:09:41 EST 2018
```

A related command is `cal` which, by default, displays a calendar of the current month.

```
[me@linuxbox ~]$ cal
     March 2018
Su Mo Tu We Th Fr Sa
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

To see the current amount of free space on our disk drives, enter `df`.

```
[me@linuxbox ~]$ df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/sda2             15115452   5012392   9949716  34% /
/dev/sda5             59631908  26545424  30008432  47% /home
/dev/sda1               147764     17370    122765  13% /boot
tmpfs                   256856         0    256856   0% /dev/shm
```

Likewise, to display the amount of free memory, enter the `free` command.

```
[me@linuxbox ~]$ free
             total       used       free     shared    buffers     cached
Mem:    513712     503976       9736          0       5312     122916
-/+ buffers/cache: 375748     137964
Swap:  1052248     104712     947536
```

Ending a Terminal Session

We can end a terminal session by either closing the terminal emulator window, by entering the `exit` command at the shell prompt, or pressing `Ctrl-d`.

```
[me@linuxbox ~]$ exit
```

The Console Behind the Curtain

Even if we have no terminal emulator running, several terminal sessions continue to run behind the graphical desktop., We can access these sessions, called *virtual terminals* or *virtual consoles*, by pressing `Ctrl-Alt-F1` through `Ctrl-Alt-F6` on most Linux distributions. When a session is accessed, it presents a login prompt into which we can enter our username and password. To switch from one virtual console to another, press `Alt-F1` through `Alt-F6`. On most system we can return to the graphical desktop by pressing `Alt-F7`.

Summing Up

This chapter marks the beginning of our journey into the Linux command line with an introduction to the shell and a brief glimpse at the command line and a lesson on how to

start and end a terminal session. We also saw how to issue some simple commands and perform a little light command line editing. That wasn't so scary was it?

In the next chapter, we'll learn a few more commands and wander around the Linux file system.

Further Reading

- To learn more about Steve Bourne, father of the Bourne Shell, see this Wikipedia article:
  http://en.wikipedia.org/wiki/Steve_Bourne
- This Wikipedia article is about Brian Fox, the original author of `bash`:
  https://en.wikipedia.org/wiki/Brian_Fox_(computer_programmer)
- Here is an article about the concept of shells in computing:
  http://en.wikipedia.org/wiki/Shell_(computing)

2 – Navigation

The first thing we need to learn (besides how to type) is how to navigate the file system on our Linux system. In this chapter we will introduce the following commands:

- `pwd` – Print name of current working directory
- `cd` – Change directory
- `ls` – List directory contents

Understanding the File System Tree

Like Windows, a Unix-like operating system such as Linux organizes its files in what is called a *hierarchical directory structure*. This means they are organized in a tree-like pattern of *directories* (sometimes called folders in other systems), which may contain files and other directories. The first directory in the file system is called the *root directory*. The root directory contains files and subdirectories, which contain more files and subdirectories and so on.

Note that unlike Windows, which has a separate file system tree for each storage device, Unix-like systems such as Linux always have a single file system tree, regardless of how many drives or storage devices are attached to the computer. Storage devices are attached (or more correctly, *mounted*) at various points on the tree according to the whims of the *system administrator*, the person (or people) responsible for the maintenance of the system.

The Current Working Directory

Most of us are probably familiar with a graphical file manager which represents the file system tree as in Figure 1. Notice that the tree is usually shown upended, that is, with the root at the top and the various branches descending below.

However, the command line has no pictures, so to navigate the file system tree we need to think of it in a different way.

Imagine that the file system is a maze shaped like an upside-down tree and we are able to

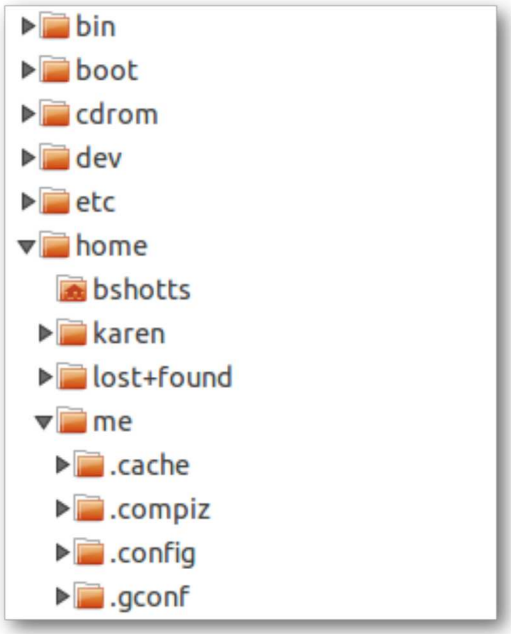

Figure 1: File system tree as shown by a graphical file manager

stand in the middle of it. At any given time, we are inside a single directory and we can see the files contained in the directory and the pathway to the directory above us (called the *parent directory*) and any subdirectories below us. The directory we are standing in is called the *current working directory*. To display the current working directory, we use the `pwd` (print working directory) command.

```
[me@linuxbox ~]$ pwd
/home/me
```

When we first log in to our system (or start a terminal emulator session) our current working directory is set to our *home directory*. Each user account is given its own home directory and it is the only place a regular user is allowed to write files.

Listing the Contents of a Directory

To list the files and directories in the current working directory, we use the `ls` command.

```
[me@linuxbox ~]$ ls
```

```
Desktop  Documents  Music  Pictures  Public  Templates  Videos
```

Actually, we can use the `ls` command to list the contents of any directory, not just the current working directory, and there are many other fun things it can do as well. We'll spend more time with `ls` in the next chapter.

Changing the Current Working Directory

To change our working directory (where we are standing in our tree-shaped maze) we use the `cd` command. To do this, type `cd` followed by the *pathname* of the desired working directory. A pathname is the route we take along the branches of the tree to get to the directory we want. We can specify pathnames in one of two different ways; as *absolute pathnames* or as *relative pathnames*. Let's deal with absolute pathnames first.

Absolute Pathnames

An absolute pathname begins with the root directory and follows the tree branch by branch until the path to the desired directory or file is completed. For example, there is a directory on our system in which most of our system's programs are installed. The directory's pathname is `/usr/bin`. This means from the root directory (represented by the leading slash in the pathname) there is a directory called "usr" which contains a directory called "bin".

```
[me@linuxbox ~]$ cd /usr/bin
[me@linuxbox bin]$ pwd
/usr/bin
[me@linuxbox bin]$ ls
...Listing of many, many files ...
```

Now we can see that we have changed the current working directory to `/usr/bin` and that it is full of files. Notice how the shell prompt has changed? As a convenience, it is usually set up to automatically display the name of the working directory.

Relative Pathnames

Where an absolute pathname starts from the root directory and leads to its destination, a relative pathname starts from the working directory. To do this, it uses a couple of special notations to represent relative positions in the file system tree. These special notations are "." (dot) and ".." (dot dot).

The "." notation refers to the working directory and the ".." notation refers to the working

directory's parent directory. Here is how it works. Let's change the working directory to `/usr/bin` again.

```
[me@linuxbox ~]$ cd /usr/bin
[me@linuxbox bin]$ pwd
/usr/bin
```

Now let's say that we wanted to change the working directory to the parent of `/usr/bin` which is `/usr`. We could do that two different ways, either using an absolute pathname.

```
[me@linuxbox bin]$ cd /usr
[me@linuxbox usr]$ pwd
/usr
```

or, using a relative pathname.

```
[me@linuxbox bin]$ cd ..
[me@linuxbox usr]$ pwd
/usr
```

Two different methods with identical results. Which one should we use? The one that requires the least typing!

Likewise, we can change the working directory from `/usr` to `/usr/bin` in two different ways, either using an absolute pathname:

```
[me@linuxbox usr]$ cd /usr/bin
[me@linuxbox bin]$ pwd
/usr/bin
```

or, using a relative pathname.

```
[me@linuxbox usr]$ cd ./bin
[me@linuxbox bin]$ pwd
/usr/bin
```

Now, there is something important to point out here. In almost all cases, we can omit the "./". It is implied. Typing:

```
[me@linuxbox usr]$ cd bin
```

does the same thing. In general, if we do not specify a pathname to something, the working directory will be assumed.

Some Helpful Shortcuts

In Table 2-1 we see some useful ways the current working directory can be quickly changed.

Table 2-1: `cd` Shortcuts

Shortcut	Result
`cd`	Changes the working directory to your home directory.
`cd -`	Changes the working directory to the previous working directory.
`cd ~`*`user_name`*	Changes the working directory to the home directory of *user_name*. For example, `cd ~bob` will change the directory to the home directory of user “bob.”

Important Facts About Filenames

On Linux systems, files are named in a manner similar to other systems such as Windows, but there are some important differences.

1. Filenames that begin with a period character are hidden. This only means that `ls` will not list them unless you say `ls -a`. When your account was created, several hidden files were placed in your home directory to configure things for your account. In Chapter 11 we will take a closer look at some of these files to see how you can customize your environment. In addition, some applications place their configuration and settings files in your home directory as hidden files.

2. Filenames and commands in Linux, like Unix, are case sensitive. The filenames “File1” and “file1” refer to different files.
3. Linux has no concept of a “file extension” like some other operating systems. You may name files any way you like. The contents and/or purpose of a file is determined by other means. Although Unix-like operating systems don’t use file extensions to determine the contents/purpose of files, many application programs do.
4. Though Linux supports long filenames that may contain embedded spaces and punctuation characters, limit the punctuation characters in the names of files you create to period, dash, and underscore. *Most importantly, do not embed spaces in filenames.* If you want to represent spaces between words in a filename, use underscore characters. You will thank yourself later.

Summing Up

This chapter explained how the shell treats the directory structure of the system. We learned about absolute and relative pathnames and the basic commands that we use to move about that structure. In the next chapter we will use this knowledge to go on a tour of a modern Linux system.